Ludwig-Maximilians-Universität München
Institut für Informatik
PD. Dr. Peer Kröger
Johannes Niedermayer, Andreas Züfle

München, 12.12.2012

Einführung in die Programmierung

WS 2012/13

Übungsblatt 8: Imperative Algorithmen

Besprechung: 19./21.12.2012 und 07.01.2013

Ende der Abgabefrist: Dienstag, 18.12.2012 14:00 Uhr.

Hinweise zur Abgabe:
Geben Sie bitte Ihre gesammelten Lösungen zu diesem Übungsblatt in einer Datei `loesung08.zip` unter `https://uniworx.ifi.lmu.de` ab.

Aufgabe 8-1 *Arrays* 3+2 Punkte

(a) Schreiben Sie in einer Datei `Medoid.java` eine Methode `medoid`, die für ein Array vom Typ `double[]` diejenige Zahl zurückgibt, die dem Mittelwert aller Zahlen am nächsten liegt.

(b) Erweitern Sie Ihr Programm um eine entsprechende `main`- Methode, so dass zum Testen obiger Methode `medoid` beim Aufruf der Klasse `Medoid` über die Kommandozeile eine Reihe von Zahlen übergeben werden kann und das Ergebnis der Methode `medoid` auf die Kommandozeile ausgegeben wird.

Beispiel: Der Aufruf
```
java Medoid 2.4 5.1 7 4 14.0 8
```
gibt
```
7.0
```
auf der Kommandozeile aus.

Hinweis: In der Klasse `java.lang.Double` finden Sie eine geeignete Methode, um einen `String` in die entsprechende Zahl vom Typ **`double`** zu konvertieren.

Sie können auch auf die API-Spezifikation zurückgreifen: http://docs.oracle.com/javase/7/docs/api/index.html.

Aufgabe 8-2 *Sieb des Eratosthenes* 0 Punkte

Nach Eratosthenes, einem Mathematiker des alten Griechenlands, wird folgender Algorithmus zur Bestimmung aller Primzahlen bis zu einer bestimmten Zahl „Sieb des Eratosthenes“ genannt:

Angenommen, Sie wollen alle Primzahlen ≤ 1000 bestimmen.

- Erstellen Sie ein Array vom Typ **`boolean`**`[]` der Länge 1001.
- Die Elemente mit Index 0 und 1 sollen den Wert **`false`** haben, alle anderen zunächst den Wert **`true`**. (0 und 1 sind *per definitionem* keine Primzahlen.)

- Fangen Sie nun an, das Array von vorne an nach Elementen vom Wert **`true`** zu durchsuchen. Für jedes solche Element a an Index i_a setzen Sie den Wert aller Elemente b, deren Index i_b ein ganzzahliges Vielfaches von i_a ist, auf den Wert **`false`**.
- Wenn Sie die Suche beendet haben, sind die Indizes mit Wert **`true`** die Primzahlen ≤ 1000.

(a) Schreiben Sie in einer Klasse `Primzahlen` eine Methode
`public static boolean``[] siebDesEratosthenes(`**`int`**` n)`,
die für eine natürliche Zahl $n > 0$ die oben beschriebene Methode implementiert.

(b) Ergänzen Sie die Klasse um eine Methode
`public static void` `printPrimzahlen(`**`int`**` n)`,
die für eine natürliche Zahl $n > 0$ die Primzahlen $\leq n$ auf die Konsole ausgibt.

(c) Ergänzen Sie die Klasse um eine `main`-Methode, die für eine beim Aufruf der Klasse (als String) angegebene natürliche Zahl > 0 die Methode `printPrimzahlen` mit dem entsprechenden **`int`**-Wert als Parameter aufruft.

Aufgabe 8-3 *Primfaktorisierung* **6+4+0+2 Punkte**

Die Primfaktorzerlegung gibt für eine Zahl $n \in \mathbb{N}$ die Multimenge ihrer Primfaktoren an, d.h. eine Multimenge P von Faktoren $p_i \in P$ so, dass gilt:

1. $n = \prod_{p_i \in P} p_i$
2. Für alle $p_i \in P$ gilt: p_i ist eine Primzahl.

Die Zahl 12 z.B. lässt sich so faktorisieren: $12 = 2 \cdot 2 \cdot 3$. Diese Zerlegung in Primfaktoren ist eindeutig.

Ein einfacher Algorithmus hierzu testet für einen Teiler p, ob die Zahl n durch p teilbar ist. Wenn das der Fall ist, ist p der erste Primfaktor von n. Anschließend werden die Primfaktoren von $\frac{n}{p}$ bestimmt. Andernfalls wird als Teiler $p+1$ getestet, solange $p+1$ nicht größer ist als n. Der erste Teiler, der getestet wird, ist 2, die kleinste Primzahl.

(a) Implementieren Sie diesen Algorithmus in einer statischen Methode `printPrimfaktoren` in einer Datei `Primfaktoren.java`, die für eine gegebene Zahl vom Typ **`int`** einen String auf die Kommandozeile ausgibt, der die Primfaktorisierung der Zahl als arithmetischen Ausdruck darstellt.

Beispiel: Wird die Methode mit dem Parameter `12` aufgerufen, dann soll sie `12 = 2 * 2 * 3` ausgeben.

(b) Unter Verwendung Ihrer Implementierung der Aufgabe 8-2 können Sie leicht eine Methode `printGesiebtePrimfaktoren` implementieren, die das gleiche Ergebnis liefert wie `printPrimfaktoren`, aber als Teiler nur Primzahlen testet.

(c) Überlegen Sie, welche der beiden Methoden effizienter ist (also weniger Speicherplatz und/oder Rechenzeit) benötigt. Theoretische Grundlagen hierzu folgen in der Vorlesung zu einem späteren Zeitpunkt, aber wenn Sie hierzu bereits Ideen haben, können Sie diese in die Dokumentation der Klasse einfließen lassen.

(d) Ergänzen Sie die Klasse `Primfaktoren` um eine `main`-Methode, die die Methoden `printPrimfaktoren` und `printGesiebtePrimfaktoren` mit einer Zahl aufruft, die als Parameter über die Kommandozeile (als String) angegeben wird. Um die Ergebnisse aus Teilaufgaben (a) und (b) zu erreichen, kann man dann also die Klasse aufrufen wie folgt:
`java Primfaktoren 12`

Aufgabe 8-4 *Addition von natürlichen Zahlen in p-adischer Darstellung* **0 Punkte**

Die Dokumentation der Klasse `Addition`, die Sie auf der Vorlesungswebseite finden, spezifiziert bereits, was diese Klasse leisten soll. Sie müssen die Datei nur entsprechend ergänzen:

(a) Ergänzen Sie den Rumpf der Methode `alphabet`, so dass das Array `ALPHABET` für das Alphabet mit den Zeichen `0,...,9,A,...,Z,a,...,z` geschickt initialisiert wird.

(b) Für die Addition werden zwei Zahlen in p-adischer Darstellung als Arrays der entsprechenden Zeichen erwartet sowie eine Basis p (in Dezimalschreibweise) der Darstellung. Das Ergebnis soll die Summe der Zahlen wiederum als Array der Zeichen (mit Basis p) ohne führende Nullen sein. Ergänzen Sie die Methode `summe`, so dass Sie diese Aufgabe erfüllt.